WARNOCK HIDDEN LINE ALGORITHM
IMPLEMENTED IN SIMULA 67

A graduate project submitted in partial satisfaction of the requirements for the degree of Master of Science in Computer Science by Robert Andrew Daymo

June, 1980
The Project of Robert Andrew Daymo is approved:

Fred Gruenberger

Ray F. Pettit

Russell J. Abbott

California State University, Northridge
# TABLE OF CONTENTS

**ABSTRACT**

1.0 Introduction

2.0 Background

2.1 Basic Concepts

2.2 Representation of a Model

2.3 Display of a Model

2.4 Depth Determining Hidden Surface Algorithms

3.0 Warnock Hidden Line Algorithm

3.1 World Model

3.2 Required User Input Data

3.3 Warnock Algorithm Processing

4.0 Verification and Modifications

4.1 Initial Verification

4.2 Window Look-ahead Modification

4.3 Edge Display Modification

4.4 Multi-edge Display Modification

4.5 Final Verification

References

Appendix 1 SIMULA Program

Appendix 2 Warnock Hidden Line Algorithm Procedure Descriptions

Table 1 Input Data Value Limits

Figure 1 Wire Frame Cube

Figure 2 Opaque Cube

Figure 3 Two Views of an Opaque Cube

Figure 4 Unit Scan Representation

Figure 5 Six Planes of a Cube

Figure 6 From/To Edge Point Array

Figure 7 Edge Array

Figure 8 Arrays Required to Model a Square

Figure 9 Incorrect Array Model of a Square

Figure 10 Incorrect Model of a Square

Figure 11 Array Data to Model a Cube

Figure 12 Representation of a Half-Circle

Figure 13 Algorithm Input Data of a Cube

Figure 14 World Model Consisting of Two Planes

Figure 15 First Quartering of World Model

Figure 16 First Set of Displayable Edges

Figure 17 Second Set of Displayable Edges

Figure 18 Quartering of Window III of World Model

Figure 19 Third Set of Displayable Edges

Figure 20 Fourth Set of Displayable Edges

Figure 21 Fifth Set of Displayable Edges

Figure 22 Displayable Edges of World Model

Figure 23 Edge Display Modification Model

Figure 24 Multi-edge in Unit Window

Figure 25 Square over Rectangle

Figure 26 Two Squares with Common Edge
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>Three Polygons with Three Common Edges</td>
<td>35</td>
</tr>
<tr>
<td>28</td>
<td>Six Polygons with Twelve Common Edges</td>
<td>36</td>
</tr>
<tr>
<td>29</td>
<td>One Inch Square Subdivision Comparison</td>
<td>36</td>
</tr>
<tr>
<td>30</td>
<td>Multi-edges in a Window</td>
<td>38</td>
</tr>
<tr>
<td>31</td>
<td>Algorithm Execution Time Comparison</td>
<td>38</td>
</tr>
</tbody>
</table>
ABSTRACT

WARNOCK HIDDEN LINE ALGORITHM IMPLEMENTED IN SIMULA 67

by

Robert Andrew Daymo

Master of Science in Computer Science

The purpose of this graduate project was to implement the Warnock Hidden Line Algorithm described in the Principles of Interactive Computer Graphics by William M. Newman and Robert F. Sproull, in SIMULA on the CDC 3170/3300 computer. The algorithm was modified twice to decrease the execution time. The first modification deleted the processing of areas where no edges existed. The second modification stopped execution for progressively smaller windows of an area after an edge was drawn. Using the SIMULA procedure ELAPSED, the modified algorithm's execution time was determined to be a linear function of the number of polygons that are input for analysis. Therefore the execution time required for a problem containing six four sided polygons is minimally twice that of a problem containing three four sided polygons. The algorithm was also modified to allow polygons with common edges to be processed.
1.0 Introduction

This document establishes the development, performance, verification, and user requirements of the Warnock Hidden Line Algorithm. The basic algorithm is detailed in the Principles of Interactive Computer Graphics by William M. Newman and Robert F. Sproull. Additional procedures were incorporated to enhance the basic algorithm.

2.0 Background

Hidden line algorithms were developed to enable three dimensional objects and perspectives to be depicted on two dimensional displays.

This project was to implement a hidden line algorithm for a non-interactive display, the CALCOMP plotter. The Warnock Hidden Line Algorithm as detailed in Newman and Sproull was chosen for this purpose.

2.1 Basic Concepts

A polygon can be defined as a set of line segments called edges. The area enclosed by the edges is the surface of the polygon.

A cube is the combination of six polygons attached at common edges. If the cube is assumed to be constructed of wire edges a wire frame cube would be visible as shown in Figure 1.

In the wire frame cube all of the edges that make up the
Figure 1  Wire Frame Cube

Figure 2  Opaque Cube
cube are visible. Instead of constructing the cube out of wire, consider the cube to be constructed from six transparent polygons. Figure 1 again represents this model of the cube. In this case the edges and surfaces of each polygon are visible. This representation of an object is not the way we normally see objects because the surfaces we normally see are opaque instead of transparent. If the surfaces of a cube are opaque the cube would appear as in Figure 2.

In Figure 2 only the surfaces closest to the viewer are visible and all other surfaces are hidden. This is one of the views of a cube that is possible from a three dimensional graphics algorithm such as Warnock's Hidden Line Algorithm.

2.2 Representation of a Model

For an algorithm to simulate what the viewer sees from a particular viewpoint, the model must be available to the algorithm in some form for computation. The model should be represented in some type of mathematical format. A mathematical representation will allow the model to be rotated around any axis and still allow an algorithm to correctly display the image of the model from any viewpoint. As the model is rotated the mathematical representation must change accordingly to correspond to the rotation.

The model that is available to the algorithm must consist of all parts of the shape, even those that are not visible from a particular viewpoint.
Figure 3a shows one view of a three-dimensional shape. Figure 3b shows the same three-dimensional shape after a ninety-degree left rotation and turning it upside down.

Representing the three-dimensional shape in the computer as depicted in either Figure 3a or Figure 3b is not enough. The combination of both figures is required for a complete representation of this object.

2.3 Display of a Model

The ability to display a model as a wire frame or transparent surface is not very difficult. The model is simply redrawn without regard to surface depth.

An opaque model creates a problem when a realistic representation is required. The surfaces that are hidden behind other surfaces should not be drawn. An algorithm has to determine when a surface is visible. This determination is important when attempting to display a shape as in Figures 3a and 3b.

The removal of hidden surfaces and edges leads to a realistic view of a model, which is defined in three dimensions on a two-dimensional display.

2.4 Depth Determining Hidden Surface Algorithms

Three depth determining types of Hidden Surface/Line Algorithms have been developed. They are classified as
Figure 3 Two Views of an Opaque Model
1) surface/surface test, 2) point/surface test, and 3) scan line test.

The Warnock Algorithm is a surface/surface test. A surface is compared to all of the other surfaces of a model. If the surface is closer to the viewer then the surface is visible. If the surface is behind another surface it is not visible to the viewer.

In the point/surface test algorithm a point of a surface is compared with another surface. If the test point is closer to the viewer than the surface then the test point's surface is visible. If the test point is behind the surface then the test point's surface is not visible.

In the scan line test algorithm the surface with the smallest \( Z \) coordinate is visible. The entire world is broken down into many scans. A scan is a unit of the entire display, see Figure 4.

3.0 Warnock Hidden Line Algorithm

3.1 World Model

For the Warnock Hidden Line Algorithm to display a cube as in Figure 2 it is necessary to represent the cube in \( X, Y, Z \) coordinates.

The cube model as represented in the computer will consist of the \( X, Y, Z \) coordinates of each edge point, the edge that connects two edge points, and the order that the edges are connected.

The cube as shown in Figure 5 will be the model that is stored in the computer. Figure 5 shows the six polygons that
Figure 4 Unit Scan Representation

Figure 5 Six Planes of a Cube
comprise the cube. Each of the edges and edge points are numbered.

The cube model consists of 24 edge points and 24 edges. Each edge point has an \(X, Y, Z\) coordinate. Using the model of Figure 5 the following data must be recorded in arrays:

1. \(X_i\) coordinate
2. \(Y_i\) coordinate
3. \(Z_i\) coordinate

The maximum number of edge points used in a model is restricted to the length of the \(X, Y, Z\) edge point arrays.

In the polygon "Front" of Figure 5 it can be seen that there are four edge points and four edges that must be represented in the computer. It has previously been stated that the \(X, Y, Z\) coordinates of each edge point is used to define the polygon. Now it will be shown how the edge points are connected to form the polygon.

Edge 1 (E1) is defined as the line connecting edge points 1 and 2. The remaining three edges can be defined using their respective edge points. Another pair of arrays will be needed to store the order of the edge points that make up an edge. Figure 6 shows the arrays and the data that would be stored.

Along with the From/To edge point arrays, an array of the edges that comprise the polygon must be built. This array is described in Figure 7.

Figure 8 shows the arrays and the respective values required
<table>
<thead>
<tr>
<th>From Edge Point Array</th>
<th>To Edge Point Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 6 From/To Edge Point Array**

**Edge Array**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Edge 1 of Polygon 1 (E1)</td>
</tr>
<tr>
<td>2</td>
<td>Edge 2 of Polygon 1 (E2)</td>
</tr>
<tr>
<td>3</td>
<td>Edge 3 of Polygon 1 (E3)</td>
</tr>
<tr>
<td>4</td>
<td>Edge 4 of Polygon 1 (E4)</td>
</tr>
</tbody>
</table>

**Figure 7 Edge Array**
### Figure 8 Arrays Required to Model a Square

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>From Edge Point</th>
<th>To Edge Point</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
<td>Z1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>X2</td>
<td>Y2</td>
<td>Z2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>X3</td>
<td>Y3</td>
<td>Z3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>X4</td>
<td>Y4</td>
<td>Z4</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

### Figure 9 Incorrect Array Model of a Square

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>From Edge Point</th>
<th>To Edge Point</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
<td>Z1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>X2</td>
<td>Y2</td>
<td>Z2</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>X3</td>
<td>Y3</td>
<td>Z3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>X4</td>
<td>Y4</td>
<td>Z4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 10 Incorrect Model of a Square
to model a simple square.

The From/To edge arrays must be consistent with the model that is being represented. If the data for the three arrays was as described in Figure 9 the object of Figure 10 would be represented instead of a square. Therefore it is important that the data be arranged in the array in a specific order to produce the desired object.

Figure 11 shows the data required to model the cube of Figure 5. The length of each array is dependent upon the complexity of the objects.

The Warnock Hidden Line Algorithm requires a minimum of three edge points and three edges to define a surface. Therefore single edge points, two edge points with a single edge, or three edge points and two edges are not allowed as objects to be included in the represented model.

The representation of a circle or any arc of a circle will require the user to define the circle to consist of a very large number of edge points and edges. Figure 12 shows how an arc of 180° could be represented with various numbers of edge points and edges. It is evident that the more edges used, the better approximation of an arc is attained. Also, the number of edges used at one radius may not be sufficient for a larger radius. Therefore the representation of any curved object will require extensive preliminary work by the user.
3.2 Required User Input Data

The user of the Warnock Hidden Line Algorithm must first reduce the model into its basic components as shown in Figure 5 and Figure 11. With this information the input for the algorithm can be prepared for the input medium. The input medium in this case is the keypunched card. The data required by the algorithm must be input in a specific order to be interpreted correctly. If this order is not followed the model that the algorithm perceives will be different from that of the user. Figure 13 shows the data that must be used to represent the object of Figure 5.

The number of polygons input for this version of the algorithm has been limited, see Table 1. The arrays that have been assigned to receive the input data will allow a maximum of fifteen three sided polygons or two cubes. The array sizes could be lengthened if a larger world with more polygons is to be entered.

3.3 Warnock Algorithm Processing

The algorithm consists of three procedures with eight subprocedures. The first procedure executed (MAIN) reads the prepared keypunched data and stores the data into the appropriate array. The second procedure (INITPOYLGONS) ranks the polygons according to their respective depths. The third procedure (WARNOCK) determines whether or not a surface is visible. If the surface is visible then the surface edge is output for display. The procedures and subprocedures are detailed in Appendix 2.
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>From Edge Point</th>
<th>To Edge Point</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
<td>Z1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>X2</td>
<td>Y2</td>
<td>Z2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>X3</td>
<td>Y3</td>
<td>Z3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>X4</td>
<td>Y4</td>
<td>Z4</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>X5</td>
<td>Y5</td>
<td>Z5</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>X6</td>
<td>Y6</td>
<td>Z6</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>X7</td>
<td>Y7</td>
<td>Z7</td>
<td>7</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>X8</td>
<td>Y8</td>
<td>Z8</td>
<td>8</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>X9</td>
<td>Y9</td>
<td>Z9</td>
<td>9</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>X10</td>
<td>Y10</td>
<td>Z10</td>
<td>10</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>X11</td>
<td>Y11</td>
<td>Z11</td>
<td>11</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>X12</td>
<td>Y12</td>
<td>Z12</td>
<td>12</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>X13</td>
<td>Y13</td>
<td>Z13</td>
<td>13</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>X14</td>
<td>Y14</td>
<td>Z14</td>
<td>14</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>X15</td>
<td>Y15</td>
<td>Z15</td>
<td>15</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>X16</td>
<td>Y16</td>
<td>Z16</td>
<td>16</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>X17</td>
<td>Y17</td>
<td>Z17</td>
<td>17</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>X18</td>
<td>Y18</td>
<td>Z18</td>
<td>18</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>X19</td>
<td>Y19</td>
<td>Z19</td>
<td>19</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>X20</td>
<td>Y20</td>
<td>Z20</td>
<td>20</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>X21</td>
<td>Y21</td>
<td>Z21</td>
<td>21</td>
<td>22</td>
<td>21</td>
</tr>
<tr>
<td>X22</td>
<td>Y22</td>
<td>Z22</td>
<td>22</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>X23</td>
<td>Y23</td>
<td>Z23</td>
<td>23</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>X24</td>
<td>Y24</td>
<td>Z24</td>
<td>24</td>
<td>21</td>
<td>24</td>
</tr>
</tbody>
</table>

Figure 11  Array Data to Model a Cube
Figure 12 Representations of a Half-Circle
### Figure 13 Algorithm Input Data of a Cube

<table>
<thead>
<tr>
<th>INPUT DATA</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>Number of points in model</td>
</tr>
<tr>
<td>24</td>
<td>Number of edges in model</td>
</tr>
<tr>
<td>6</td>
<td>Number of polygons in model</td>
</tr>
<tr>
<td>X1, Y1, Z1</td>
<td>&quot;Front&quot; polygon edge coordinates</td>
</tr>
<tr>
<td>X2, Y2, Z2</td>
<td>&quot;Back&quot; polygon edge coordinates</td>
</tr>
<tr>
<td>X3, Y3, Z3</td>
<td>&quot;Side 2&quot; polygon edge coordinates</td>
</tr>
<tr>
<td>X4, Y4, Z4</td>
<td>&quot;Side 1&quot; polygon edge coordinates</td>
</tr>
<tr>
<td>X5, Y5, Z5</td>
<td>&quot;Top&quot; polygon edge coordinates</td>
</tr>
<tr>
<td>X6, Y6, Z6</td>
<td>&quot;Bottom&quot; polygon edge coordinates</td>
</tr>
<tr>
<td>X7, Y7, Z7</td>
<td>&quot;From&quot; &quot;To&quot; edge point pairs for polygon &quot;Front&quot;</td>
</tr>
<tr>
<td>X8, Y8, Z8</td>
<td>&quot;From&quot; &quot;To&quot; edge point pairs for polygon &quot;Back&quot;</td>
</tr>
<tr>
<td>X9, Y9, Z9</td>
<td>&quot;From&quot; &quot;To&quot; edge point pairs for polygon &quot;Side 2&quot;</td>
</tr>
<tr>
<td>X10, Y10, Z10</td>
<td>&quot;From&quot; &quot;To&quot; edge point pairs for polygon &quot;Side 1&quot;</td>
</tr>
<tr>
<td>X11, Y11, Z11</td>
<td>&quot;From&quot; &quot;To&quot; edge point pairs for polygon &quot;Top&quot;</td>
</tr>
<tr>
<td>X12, Y12, Z12</td>
<td>&quot;From&quot; &quot;To&quot; edge point pairs for polygon &quot;Bottom&quot;</td>
</tr>
<tr>
<td>X13, Y13, Z13</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X14, Y14, Z14</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X15, Y15, Z15</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X16, Y16, Z16</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X17, Y17, Z17</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X18, Y18, Z18</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X19, Y19, Z19</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X20, Y20, Z20</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X21, Y21, Z21</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X22, Y22, Z22</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X23, Y23, Z23</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
<tr>
<td>X24, Y24, Z24</td>
<td>4, 12, 11, 14, 15, 16, 17</td>
</tr>
</tbody>
</table>

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22, 23, 24, 25
**Figure 13 (cont.)**

<table>
<thead>
<tr>
<th>INPUT DATA</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Number of edges in polygon &quot;Top&quot;</td>
</tr>
<tr>
<td>17,18,19,20</td>
<td>Edge order for polygon &quot;Top&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Number of edges in polygon &quot;Bottom&quot;</td>
</tr>
<tr>
<td>21,22,23,24</td>
<td>Edge order for polygon &quot;Bottom&quot;</td>
</tr>
<tr>
<td>Data</td>
<td>Min Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Number of points</td>
<td>3</td>
</tr>
<tr>
<td>Number of edges</td>
<td>3</td>
</tr>
<tr>
<td>Number of polygons</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>+1</td>
</tr>
<tr>
<td>Edge connect</td>
<td>1</td>
</tr>
<tr>
<td>Number of edges in this polygon</td>
<td>3</td>
</tr>
<tr>
<td>Edge order</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1  Input Data Value Limits
Procedure WARNOCK is a recursive procedure. WARNOCK calls itself until all processing has been completed. For each surface that is visible, the edges that make up the surface are output for display.

The algorithm works on a windowing process. The world that the algorithm sees is defined by a window. In this case the window is 1024 x 1024 units, see Figure 14.

Two parallel planes are placed within this window. Plane 'abcd' is in front of plane 'efgh'. The algorithm will determine if a surface is within the current window. If a surface is not within the current window no further processing of this window is required. If a surface is within the window the algorithm will attempt to display the edges of the surface.

However, an edge is displayable only after 1) the edge has been found to be in front of all other surfaces and 2) the edge penetrates the window. In the example, Figure 14, since the second condition of the requirements to display a surface edge is not met the window is subdivided as shown in Figure 15. Here the window has been broken into four quarters. Now these four windows have to be considered for processing.

The first window to be processed, in this example, will be the window marked I. First it is necessary to determine if a surface edge is visible. The algorithm calls itself to make this determination.
The surface edges that are present in window I are labeled 'a' and 'b'. The algorithm determines that the polygon containing the edges 'a' and 'b' is the only polygon in this window. Therefore the edges are viewable and are output for display. Only the portion of the edge(s) within this window are output, see Figure 16. Since no further processing is required the algorithm then calls itself to process window II of Figure 15.

The surface edges that are present in window II of Figure 15 are labeled 'b' and 'c'. The algorithm determines that the polygon containing edges 'b' and 'c' is the only polygon in this window. Therefore the edges must be viewable and are output for display. Only the portion of the edge(s) within this window are output, see Figure 17. Since no further processing is required the algorithm calls itself again to process window III of Figure 15.

The surface edges that are present in window III of Figure 15 are labeled 'a' and 'd' for one polygon and 'e', 'f', and 'h' for the second polygon. The algorithm determines that more than one polygon has edges that are viewable and penetrate the window. Therefore window III of Figure 15 will be quartered as shown in Figure 18. Again the algorithm calls itself to process window I of Figure 18.

Window I of Figure 18 contains no polygons so the algorithm proceeds to window II. The surface edges in window II are labeled 'a' and 'd' for one polygon and 'e', 'f', and 'h' for the second polygon.
Figure 14 World Model Consisting of Two Planes

Figure 15 First Quartering of World Model
Figure 16 First Set of Displayable Edges

Figure 17 Second Set of Displayable Edges
Figure 18 Quartering of Window III of World Model

Figure 19 Third Set of Displayable Edges
As previously stated the polygon containing edges 'ef' is behind the polygon containing edges 'ad'. Therefore the algorithm determines that the polygon containing edges 'a' and 'd' is the only polygon in this window. The edges must therefore be viewable and are output for display. Only the portion of the edge(s) within this window are output, Figure 19.

Since no further processing of window II of Figure 18 is required the algorithm calls itself to process window III. Window III contains no polygons so the algorithm proceeds to window IV of Figure 18.

The surface edges that are present in window IV of Figure 18 are labeled 'e' and 'h'. The algorithm determines that the polygon containing edges 'e' and 'h' is the only polygon in this window. The edges must therefore be viewable and are output for display. Only the portion of the edges within this window are output as in Figure 20.

Since no further processing of window IV of Figure 18 is required the algorithm calls itself to process window IV of Figure 15. The processing of this window is similar to the processing of window III of Figure 15. The displayable output for this window is shown in Figure 21.

Figure 22 shows the resultant picture after combining the outputs shown in Figures 16 through 21.

The procedure described in this section is the windowing
Figure 20  Fourth Set of Displayable Edges

Figure 21  Fifth Set of Displayable Edges
Figure 22 Displayable Edges of World Model
process that is used in the implemented WARNOCK algorithm. The algorithm as described in Newman and Sproull quarters each window until the window size is one unit. The algorithm does not stop the quartering process if nothing exists in the window. This is one of the improvements that has been made in the algorithm that will be discussed later.

4.0 Verification and Modifications

The Warnock Hidden Line Algorithm as detailed in Newman and Sproull was converted to SIMULA. Verification of the algorithm was done using the two polygons depicted in Figure 25. Modifications to the algorithm were required to produce execution times under five minutes for situations as shown in Figures 25, 26, and 27.

4.1 Initial Verification

The two polygon world of Figure 25 was used to verify the algorithm. A 1024 unit square window is used to view and display the world model. The algorithm does consecutive quartering of a window, until the window size becomes one unit. The number of windows processed per window size is

<table>
<thead>
<tr>
<th>#windows</th>
<th>window size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1024</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>1024</td>
<td>32</td>
</tr>
<tr>
<td>4096</td>
<td>16</td>
</tr>
<tr>
<td>16384</td>
<td>8</td>
</tr>
<tr>
<td>65536</td>
<td>4</td>
</tr>
</tbody>
</table>
The initial modification was a parameter change. The algorithm was changed to not process window sizes of eight units or less. As the number of recursive calls was reduced the execution time was reduced. When the smallest window size was sixteen units, the execution time for Figure 25 was approximately nine minutes.

There were discernible gaps at the edge intersection of the two polygons. To remove the gaps and further reduce execution time more modifications were required.

4.2 Window Look-ahead Modification

The procedure NOEDGE was developed to determine if an edge of any polygon is in the next window area to be evaluated i.e. look-ahead and check if the next window must be processed. A simple approach was taken to determine the status (edge intersections) of the window. If all the edges of all the polygons are to the right, left, top, or bottom of the window, the window is marked as not requiring further processing. If the first edge that passes through the window, is enclosed by the window, or could possibly pass through the window because the end points were not to the right, left, top, or bottom of the window, the window is marked as requiring further processing.

The addition of this procedure reduced significantly the number of windows that required processing. Before the incorporation
of NOEDGE, 1 398 100 subdivisions of the display surface were required before the algorithm execution was terminated. Upon incorporating NOEDGE, the amount of subdivisions for a simple one inch square polygon was reduced to 392 subdivisions of the display surface.

Three hundred seventy seven windows of various sizes were found to contain no pertinent data and did not require processing. It should be noted that the number of windows processed and those checked for pertinent data increase as the size of the polygon and as the number of polygons in the display surface increases.

The addition of NOEDGE greatly reduced the execution time for Figure 25. With the reduction in time, the minimum window size was changed to two units. The discernible gaps between the polygon edges disappeared. The extra resolution attained by a window size of one unit compared to two units was not discernible on the CALCOMP plotter. Since the quality of the result was not effected, the minimum window size of two units was maintained.

The execution time for Figure 25 with a minimum window size of two units was less than five minutes. With the addition of NOEDGE not only was the execution time reduced but also the quality of the final output was enhanced.

4.3 Edge Display Modification

The original algorithm repetitively drew portions of an edge.
Figure 23 shows a portion of the world with the window size being eight units.

In this example the edge 'ab' or portions of edge 'ab' would be drawn at the following window sizes:

<table>
<thead>
<tr>
<th>edge</th>
<th>window size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>8</td>
</tr>
<tr>
<td>ad</td>
<td>4</td>
</tr>
<tr>
<td>di</td>
<td>4</td>
</tr>
<tr>
<td>ib</td>
<td>4</td>
</tr>
<tr>
<td>ad</td>
<td>2</td>
</tr>
<tr>
<td>de</td>
<td>2</td>
</tr>
<tr>
<td>eh</td>
<td>2</td>
</tr>
<tr>
<td>hi</td>
<td>2</td>
</tr>
<tr>
<td>ib</td>
<td>2</td>
</tr>
<tr>
<td>ac</td>
<td>1</td>
</tr>
<tr>
<td>cd</td>
<td>1</td>
</tr>
<tr>
<td>de</td>
<td>1</td>
</tr>
<tr>
<td>ef</td>
<td>1</td>
</tr>
<tr>
<td>fg</td>
<td>1</td>
</tr>
<tr>
<td>gh</td>
<td>1</td>
</tr>
<tr>
<td>hi</td>
<td>1</td>
</tr>
<tr>
<td>ij</td>
<td>1</td>
</tr>
<tr>
<td>jb</td>
<td>1</td>
</tr>
</tbody>
</table>

The time to process portions of an already displayed edge was wasting processor and CALCOMP plotter time. The algorithm was modified to draw the first encounter of an edge and stop the windowing process in this area. In this example the edge 'ab' would be drawn at a window size of eight units. No further processing would occur in this window area.

The algorithm was modified by the addition of a variable to indicate when one and only one polygon's edge passes through the window for display. In Figure 23 assume edge 'ab' belongs to a single polygon. When this window is processed the variable is set to indicate
Figure 23  Edge Display Modification Model
that no further process of this window area is required because only this single edge exists in this window. This modification works in conjunction with the procedure NOEDGE described in Section 4.2. It can be thought of as an extension of the NOEDGE procedure.

The amount of subdivisions for a simple one inch square polygon was reduced as shown in Figure 29. The number of windows processed and those checked for pertinent data increase as the size of the polygon increases but not as rapidly as in the case of the loo-ahead procedure without this modification.

The execution time for Figure 25 was reduced to less than two minutes with the addition of this modification.

4.4 Multi-edge Display Modification

The algorithm would not display an edge if more than one edge would intersect a window. Figure 24 is an exaggerated example of two edges occupying the same window. The edges in Figure 24 represent a portion of the common edge of Figure 26. The separation of the two edges is for pictorial representation only. As depicted in Figure 26 the two edges are colinear.

A modification was required to display polygons as shown in Figures 26, 27, and 28. To process a common edge and multiple edges intersecting a window, two routines were incorporated.

The first routine is executed after the original algorithm has decided that two or more edges from more than one polygon intersect
Figure 24 Multi-edge in Unit Window

Figure 25 Square over Rectangle
the window. The routine determines if the edges that intersect the window are common as in Figure 26 i.e. the end points are the same for both edges. If they are common then this edge is displayed and a variable is set to prevent further processing of smaller windows in this area of the display.

The second routine is executed if more than one common edge intersects the window, see Figure 30. Figure 30 is an expansion of one corner of Figure 27. The polygon that has been previously determined to be closer to the viewer has its edge(s) drawn. Assume in this example the edges 'ab' and 'bd' are drawn. The edge 'bc' is from the second polygon and this polygon may not be listed on the polygon viewable list - the list of polygons that are viewable in the present window and are to be drawn. Since this may occur a variable is not set and processing will continue in this area of the display surface. Therefore only when the second routine is executed can multiple displays of the same edge(s) occur.

The addition of this routine allowed the model depicted in Figure 27 to be displayed in less than five minutes. Without this modification Figure 27 was never processed correctly.

4.5 Final Verification

The ability of the algorithm to process the situation shown in Figure 28 was attempted. The execution time was over ten minutes. To determine the reason for such an increase in time the SIMULA procedure ELAPSED was run.
Figure 26 Two Squares with Common Edges

Figure 27 Three Polygons with Three Common Edges
Figure 28 Six Polygons with Twelve Common Edges

<table>
<thead>
<tr>
<th>Modification</th>
<th>Displayable Subdivisions</th>
<th>Non-displayable Subdivisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>392</td>
<td>377</td>
</tr>
<tr>
<td>3</td>
<td>52</td>
<td>37</td>
</tr>
</tbody>
</table>

Figure 29 One Inch Square Subdivision Comparison
ELAPSED allows the user to determine the time to process a procedure or instructions. Each procedure was timed. The call to ELAPSED was placed on the entry and exit for each procedure. Two situations were tested to get relative times for each procedure. Figures 27 and 28 show the situations tested.

The procedure MAIN (Appendix 2) was found to be a linear procedure. As the number of edges increased the execution time of MAIN increased. The amount of time to process the input data for Figure 28 is double that for Figure 27. Figure 31 shows the procedure times recorded for the two examples of Figures 27 and 28.

Assume the number of times that the WARNOCK and NOEDGE are executed is 1000 and 800 respectively for both Figures 27 and 28.

The total execution time for Figure 27 in this case is

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>0.1 sec</td>
</tr>
<tr>
<td>INITPOLYGONS</td>
<td>0.165 sec</td>
</tr>
<tr>
<td>WARNOCK</td>
<td>220.0 sec</td>
</tr>
<tr>
<td>NOEDGE</td>
<td>48.0 sec</td>
</tr>
</tbody>
</table>

268.265 sec. = 4 min. 28 sec.

The total time for Figure 28 in this case is

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>0.258 sec</td>
</tr>
<tr>
<td>INITPOLYGONS</td>
<td>0.207 sec</td>
</tr>
<tr>
<td>WARNOCK</td>
<td>470.0 sec</td>
</tr>
<tr>
<td>NOEDGE</td>
<td>96.0 sec</td>
</tr>
</tbody>
</table>

566.465 sec. = 9 min. 26 sec.

The procedure INITPOLYGONS was also found to be a linear procedure. Again the number of edges was the determining factor.

The additional time required by MAIN and INITPOLYGONS is
Figure 30 Multi-edges in a Window

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Figure 27 Time (ms)</th>
<th>Figure 28 Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>100</td>
<td>258</td>
</tr>
<tr>
<td>INITPOLYGONS</td>
<td>165</td>
<td>207</td>
</tr>
<tr>
<td>WARNOCK</td>
<td>220</td>
<td>470</td>
</tr>
<tr>
<td>NOEDGE</td>
<td>60</td>
<td>120</td>
</tr>
</tbody>
</table>

Note: The elapsed times for WARNOCK and NOEDGE are average times after twelve window subdivisions.

Figure 31 Algorithm Execution Time Comparison
insignificant when compared to the amount of time required by the
recursive procedure WARNOCK. WARNOCK is also a linear
procedure. Since WARNOCK recursively calls itself everytime
another window division is required, the total execution time is
directly related to the procedure WARNOCK.

Through the use of ELAPSED, the algorithm was found to be
linear. The feasibility of using the algorithm for computer graphic
problems does not appear beneficial because of the time factor.
REFERENCES


BEGIN
REAL ARRAY XS,YS,ZS(1:50); INTEGER ARRAY EDI,ED2,EDLINK(1:50);
REAL ARRAY POLYA,POLYA,POLYC,POLYD,POLYMIN(1:50);
INTEGER ARRAY POLYEDGE,POLYLINK,POLYLIST(1:50);
REAL W1,W2,W3,W4,W5,W6,W7,W8,W9,W10,W11,W12,W13;
XX1,XX2,YY1,YY2,ZMIN1,ZMIN2,ZMIN3,ZMIN4,ZMAX1,ZMAX2,ZMAX3,ZMAX4;
REAL LEFT,RIGHT,SIZE;
REAL EPSILON;
INTEGER P,SURROUNDFPS,INTERSECTORS,POLYPTX,INTER;
NEXTEDGE,DELTATETA,THETA,HOLD,ALTP,J;
HOLKAN PERKRATE;
INTEGER POLYNUM,EDGENUM,POLYNUMS,1,RETURN,T,K,C1,C2,T,T1;
INTEGER ARRAYLENGTH;
INTEGER [C];
REAL TIME;
EXTERNAL PROCEDURE BLOT();
EXTERNAL PROCEDURE BLOT1();
EXTERNAL PROCEDURE PL0T();

PROCEDURE MAIN; BEGIN
ARRAYLENGTH := 50;
POLYNUM := 1
EDGENUM := 1
POLYTX := 1
T := 0
WHILE 1 NE POLYNUM DO BEGIN T := T + 1;
XS(T) := TINT; YS(T) := TINT; ZS(T) := TINT;
END;
T := 0
WHILE 1 NE EDGENUM DO BEGIN T := T + 1;
END;
ED(T) := 1
FOR T := 1 STEP 1 UNTIL ARRAYLENGTH DO BEGIN POLYX(T) := -T;
END;
POLYTX := 0;
FOR T := 1 STEP 1 UNTIL POLYNUM+2 DO BEGIN IF T EQ 1 THEN HECHT := T := T + 2;
POLYTX := 1
POLYX(T) := POLYTX;
POLYTX := T
J := TINT;
POLYEDGE(T) := K := FINDEDGE;
WHILE J NE 1 DO BEGIN K := FINDEDGE(T)
END;
EDLINK(K) := 0
K := POLYEDGE(T); J := ED(K);
IF J NE ED(K) AND IF ED(K) EDLINK(K) THEN HECHT := ED(K) := ED(K) := T;
END;
WHILE K NE 0 DO BEGIN ED := EDLINK(K); IF ED(K) NE ED1(J) AND IF ED(K) EDLINK(J) THEN BEGIN ED(K) := ED1(J);
ED1(J) := ED(K);
END;
IF J NE 0 THEN K := 0
END;
K := J
END;
END;
INTEGER PROCEDURE FINDEDGE; BEGIN
INTEGER I;
I := IMINT;
IF ENLINK[I] EQ -1 THEN FINDEDGE := I ELSE BEGIN
EDGENUM := EDGENUM + 1; ED1[EDGENUM] := ED1[I];
ED2[EDGENUM] := ED2[I]; FINDEDGE := EDGENUM; END;
END;

INTEGER PROCEDURE GETNEXTEDGE; BEGIN INTEGER I;
IF NEXTEDGE EQ 0 THEN GETNEXTEDGE := 0
ELSE BEGIN I := ED1[NEXTEDGE]; PX := XS[I]; PY := YS[I];
PZ := ZS[I]; I := ED2[NEXTEDGE]; NX := XS[I]; NY := YS[I];
NZ := ZS[I]; NEXTEDGE := ED1[LINK[NEXTEDGE]]; GETNEXTEDGE := I;
END;
END;

REAL, PROCEDURE GFTZ(P,X,Y);
INTEGER P; REAL X,Y;
BEGIN
GFTZ := (((POLY[P]*X-POLY[P]*Y-POLY[P])/POLY[P]);
END;

INTEGER PROCEDURE ANGLE(X,Y);
REAL X,Y;
BEGIN INTEGER RUN;
RUN := 0;
ANGLE := 4;
IF X LT 0 THEN BEGIN IF Y GT 0 THEN ANGLE := 3;
IF Y LT 0 THEN ANGLE := 5;
RUN := 1;
END;
IF RUN EQ 0 THEN BEGIN
ANGLE := 0;
IF X GT 0 THEN BEGIN IF Y GT 0 THEN ANGLE := 1;
IF Y LT 0 THEN ANGLE := 7;
RUN := 1; END;
END;
IF RUN EQ 0 THEN BEGIN
IF Y GT 0 THEN ANGLE := 2; IF Y LT 0 THEN ANGLE := 6;
END;
END;

INTEGER PROCEDURE PUSH(AL,RT);
INTEGER AL; REAL RT;
BEGIN
INTEGER P;
IF AL EQ 0 THEN BEGIN YY1 := (YY2-YY1)*(RT-XX1)/(YY2-XX1)+YY1;
XX1 := RT; END ELSE BEGIN YY1 := (YY2-XX1)*(RT-YY1)/(YY2-YY1)+XY1;
XX1 := RT; YY1 := YY1*COFF(XX1,YY1); END;
END;
INTEGER PROCEDURE WC0DE(X, Y):
REAL X, Y; BEGIN INTEGER W;
W := 0;
IF X LT W1X THEN W := 1; IF X GT W1X THEN W := W+2;
IF Y LT W1Y THEN W := W+4; IF Y GT W1Y THEN W := W+8;
WC0DE := W;
END;

INTEGER PROCEDURE CLIP; BEGIN INTEGER C, AX, A1, A2, A3;
INTEGER C11, C22, D1, D2, D3, D4, E1, E2, E3, E4;
C := 1;
DELTA THETA := 0;
XX1 := NX; XX2 := PX; YY1 := NY; YY2 := PY;
C1 := WC0DE(XX1, YY1); C2 := WC0DE(XX2, YY2);
IF C > 1 THEN BEGIN
WHILE C1 GT 0 OR C2 GT 0 DO BEGIN
C1 := C1 + C2 := C2;
IF C1 GT X THEN BEGIN C1 := C1 - 1; D1 := 1; END ELSE D1 := 0;
IF C1 GT 4 THEN BEGIN C1 := C1 - 4; D2 := 1; END ELSE D2 := 0;
IF C1 GT 2 THEN BEGIN C1 := C1 - 2; D3 := 1; END ELSE D3 := 0;
IF C1 GT 1 THEN D4 := 1 ELSE D4 := 0;
IF C2 GT 8 THEN BEGIN C2 := C2 - 8; F1 := 1; END ELSE F1 := 0;
IF C2 GT 4 THEN BEGIN C2 := C2 - 4; F2 := 1; END ELSE F2 := 0;
IF C2 GT 2 THEN BEGIN C2 := C2 - 2; K3 := 1; END ELSE K3 := 0;
IF C2 GT 1 THEN F4 := 1 ELSE F4 := 0;
IF D1 NE 0 AND F1 NE 0 OR D2 NE 0 AND F2 NE 0 OR D3 NE 0 AND F3 NE 0 OR D4 NE 0 AND F4 NE 0 THEN BEGIN
A1 := ANGLE(NX, NY);
A2 := ANGLE(XX1, YY1); A3 := ANGLE(XX2, YY2);
A4 := A1 + A2;
IF ANGLE(A4) GT 3 THEN BEGIN IF A1 LT 0 THEN
A1 := A1 + X;
ELSE BEGIN A1 := A1 - X; END; END;
A4 := A4 + X;
IF ANGLE(A4) LT 0 THEN BEGIN
A4 := A4 - X; END; END;
DELTA THETA := A1 + A2; C := 0; END;
IF C NE 0 THEN BEGIN
IF C1 EQ 0 THEN BEGIN T := C1; C1 := C2; C2 := T;
T1 := XX1; XX1 := YY2; YY2 := T1;
T1 := YY1; YY1 := YY2; YY2 := T1;
IF C1 GT 1 THEN BEGIN C1 := C1 - 1; END;
IF C1 NE 0 THEN BEGIN
IF C1 GT 1 THEN BEGIN PUSH(1, XX1); C1 := -1; END;
END;
IF C1 GT 1 THEN BEGIN PUSH(1, YY1); C1 := -1; END;
END;
END;
IF C EQ 0 THEN C1 := C2 := 0;
END;
END;
IF C EQ 1 THEN C := 0; CLIP := C;
END;
PROCEDURE INITPOLYGONS; BEGIN
  [Boolean Change]
  REAL X1,X2,X3,Y1,Y2,Y3,Z1,Z2,Z3;
  OLDP := 0; P := POLYTR; WHILE P NE 0 DO BEGIN
    NEXTEDGE := POLYEDGE[P]; GETNEXTEDGE;
    X1 := PX; Y1 := PY; Z1 := PZ;
    X2 := PX-X1; Y2 := PY-Y1; Z2 := PZ-Z1;
    POLYC[P] := X3*Z2-Y3*Z1;
    POLYD[P] := -(POLYA[P]*X1+POLYB[P]*Y1+POLYC[P]*Z1);
    ZMIN := 0; NEXTEDGE := POLYEDGE[P];
    WHILE GETNEXTEDGE NE 0 DO BEGIN IF ZMIN GT PZ THEN
      ZMIN := PZ; END;
    END;
    IF POLYD[P] EQ 0 THEN BEGIN IF OLDP EQ 0 THEN
      POLYTR := P; END ELSE BEGIN POLYLINK[OLDP] := POLYLINK[P]; END;
    END; ELSE BEGIN OLDP := P; END;
    IF CHANGE := TRUE END;
    WHILE CHANGE DO BEGIN
      CHANGE := FALSE; OLDP := 0;
      IF P NE 0 THEN BEGIN J := POLYLINK[P];
        IF J NE 0 THEN BEGIN
          IF POLYZMIN[J] GT POLYZMIN[I] THEN BEGIN
            IF OLDP EQ 0 THEN POLYTR := J ELSE
              POLYLINK[OLDP] := J;
          END;
          CHANGE := TRUE;
          OLDP := J; END ELSE BEGIN[HERE];
          OLDP := P;
        END;
        P := POLYTR;
        THETA := 0;
        RETURN := 0;
      END;
      FOR T := 1 STEP 1 UNTIL POLYTR+1 DO BEGIN POLYLIST[T] := 0; END;
      WHILE P NE 0 DO BEGIN IF POLYZMIN[P] LE ZMINMAX THEN BEGIN
        THETA := 0; NEXTEDGE := POLYEDGE[P];
        WHILE GETNEXTEDGE NE 0 AND THETA NE -1 DO BEGIN
          THETA := POLYLIST[P] := INTERSECTORS; INTERSECTORS := P;
          THETA := -1; END;
        END; IF THETA := THETA+DELTA; THETA; END;
IF ABS(THETA) EQ 8 THEN BEGIN POLYLIST(p) := SURROUNDERS;
SURROUNDERS := p; Z1 := GETZ(p, W, X, W, Y);
Z2 := GETZ(p, W, X, W, Y); Z3 := GETZ(p, W, X, W, Y);
Z4 := GETZ(p, W, X, W, Y); IF Z2 GT Z1 THEN Z1 := Z2;
IF Z3 GT Z1 THEN Z1 := Z3; IF Z4 GT Z1 THEN Z1 := Z4;
ZMIN MAX := Z1 END;
END; P := POLYLINK(p); END;
ZMIN1 := ZMIN2 := ZMIN3 := ZMIN4 := 0; HIDER := 0;
PENETRATE := FALSE;
WHILE SURROUNDERS GT 1 DO BEGIN
Z1 := GETZ(SURROUNDERS, W, X, W, Y);
Z2 := GETZ(SURROUNDERS, W, X, W, Y);
Z3 := GETZ(SURROUNDERS, W, X, W, Y);
Z4 := GETZ(SURROUNDERS, W, X, W, Y);
IF Z1 LT ZMIN1 AND Z2 LT ZMIN2 AND Z3 LT ZMIN3 AND
Z4 LT ZMIN4 THEN BEGIN HIDER := 1; SURROUNDERS := ZMIN4;
ZMIN1 := ZMAX1 := Z1; ZMIN2 := ZMAX2 := Z2;
ZMIN3 := ZMAX3 := Z3; ZMIN4 := ZMAX4 := Z4; END ELSE BEGIN
IF Z1 LT ZMAX1 AND Z2 LT ZMAX2 AND Z3 LT ZMAX3 AND
Z4 LT ZMAX4 THEN BEGIN PENETRATE := TRUE;
IF Z1 LT ZMIN1 THEN ZMIN1 := Z1;
IF Z2 LT ZMIN2 THEN ZMIN2 := Z2;
IF Z3 LT ZMIN3 THEN ZMIN3 := Z3;
IF Z4 LT ZMIN4 THEN ZMIN4 := Z4;
IF Z1 GT ZMAX1 THEN ZMAX1 := Z1;
IF Z2 GT ZMAX2 THEN ZMAX2 := Z2;
IF Z3 GT ZMAX3 THEN ZMAX3 := Z3;
IF Z4 GT ZMAX4 THEN ZMAX4 := Z4;
END; END; SURROUNDERS := POLYLIST(SURROUNDERS); END;
IF NOT PENETRATE AND RETURN TO 0 THEN BEGIN
HIDP := 0; P := INTERSECTORS;
WHILE GT 1 AND HIDER GT 1 DO BEGIN
Z1 := GETZ(p, W, X, W, Y); Z2 := GETZ(p, W, X, W, Y);
Z3 := GETZ(p, W, X, W, Y); Z4 := GETZ(p, W, X, W, Y);
IF Z1 GT ZMAX1 AND Z2 GT ZMAX2 AND Z3 GT ZMAX3 AND
Z4 GT ZMAX4 THEN BEGIN HIDER := 1; P := POLYLIST(p);
IF nied TO 0 THEN BEGIN INTERSECTORS := 1; POLYLIST(p) := p;
END ELSE BEGIN nied := P;
IF Z1 GT ZMIN1 AND Z2 GT ZMIN2 AND Z3 GT ZMIN3 AND
Z4 GT ZMIN4 THEN BEGIN PENETRATE := TRUE;
HIDP := EBD;
END; IF HIDER NE 0 THEN := POLYLIST(p); END;
IF NOT PENETRATE THEN BEGIN IF INTERSECTORS GT 1 THEN BEGIN
NEXTEDGE := POLYLINK(INTERSECTORS);
WHILE GETEXTFIDE NE 0 AND INTERSECTORS GE 1 DO BEGIN
IF CLIP TO 0 THEN BEGIN
XY1 := XX1/100; YY1 := YY1/100; XX2 := XX2/100; YY2 := YY2/100;
POLY(xx1,yy1,xx2,yy2,1); POLY(xy1,yy1,xx2,yy2,2);
POLY(xx1,yy1,xx2,yy2,3); F := 1;
END ELSE BEGIN RETURN TO 1;
END; END; END ELSE BEGIN
P := POLYLINK; XX1 := YY1 := XX2 := YY2 := 0; PS1 := PSV := 0;
WHILE CLIP TO 0 AND RETURN TO 0 DO BEGIN
IF POLYLIST(p) NE 0 AND RETURN TO 0 THEN BEGIN
NEXTEDGE := POLYLINK(p); WHILE GETEXTFIDE NE 0 AND RETURN TO 0
DO BEGIN IF CLIP NE 0 AND RETURN FO 0 THEN BEGIN
IF PS EQ 0 THEN BEGIN PS := P; XX1S := XX1; YY1S := YY1;
XX2S := XX2; YY2S := YY2; END ELSE BEGIN
IF PS NE P THEN RETURN := 1;
IF PS NE P THEN BEGIN IF XX1S NE XX1 OR YY1S NE YY1 OR
XX2S NE XX2 OR YY2S NE YY2 THEN RETURN := 1 ELSE PSV := P;
END; END; END;
END;
F := POLYLINK(P); END; IF RETURN FO 0 AND PS NE 0 THEN BEGIN
IF PS NE 0 THEN BEGIN
P := 1; XX1 := XX1S/100; YY1 := YY1S/100; XX2 := XX2S/100;
YY2 := YY2S/100; PLOT(XX1,YY1,3); PLOT(XX1,YY1,2); PLOT(XX2,YY2,2); PLOT(XX2,YY2,3); RETURN := 1;
END; END ELSE BEGIN
PS := 0; P := POLYPTA; WHILE P NE 0 DO BEGIN
IF POLYLIST(P) NE 1 THEN BEGIN IF PS EQ 0 THEN PS := P
ELSE PS := -1; END; P := POLYLINK(P); END;
IF PS GT 0 THEN BEGIN NEXTEDGE := POLYEDGE[PS]; WHILE GETNEXTEDGE NE 0 DO BEGIN IF CLIP NE 0 THEN BEGIN
XX1 := XX1/100; YY1 := YY1/100; XX2 := XX2/100;
YY2 := YY2/100; PLOT(XX1,YY1,3); PLOT(XX1,YY1,2); PLOT(XX2,YY2,2); PLOT(XX2,YY2,3); F := F+1;
END; END; IF F GT 1 THEN F := 0; END;
END ELSE BEGIN
PENETRATE := TRUE; RETURN := 1; END;
END; END;
END;
IF PENETRATE THEN BEGIN IF RETURN FO 0 AND S EQ 2 THEN DRAW
END;
END;
IF F EQ 0 THEN BEGIN
IF S GT 2 THEN BEGIN S := S/2; S1F := S;
IF NOEDGE FO = 1 THEN WARNOCK(1,PS,S);
SIZE := S; LEFT := [1+S]; BOTTOM := R;
IF NOEDGE FO = 1 THEN WARNOCK(1+S,R,S);
SIZE := S; LEFT := [1+S]; BOTTOM := R+S;
END;
END;
END;
END;

INTEGER PROCEDURE NOEDGE; BEGIN INTEGER N;
N := 0; P := POLYPTA;
WHILE P NE 0 AND N FO 0 DO BEGIN NEXTEDGE := POLYEDGES(P);
WHILE GETNEXTEDGE NE 0 AND N FO 0 DO BEGIN N := -1;
IF MX GT LEFT AND PX LT LEFT THEN N := 0;
IF NY GT LEFT+SIZE-EPSILON AND PX GT LEFT+SIZE-EPSILON THEN N := 0;
IF NY LT BOTTOM AND PY LT BOTTOM THEN N := 0;
IF NY GT BOTTOM+SIZE-EPSILON AND PX GT BOTTOM+SIZE-EPSILON THEN N := 0; END;
P := POLYLINK(P); N := NOEDGE := N;
END;
MAIN:
INITPOLYCON:
PLOTS:
plot(0,0,3);
warnock(0,0,1024);
plot(0,0,999); plotgen;
end;
Procedure MAIN

The purpose of this procedure is to input the problem data and initialize arrays and variables that will be used by the Looker and Thinker routines (the WARNOCK procedure). The input data sequence is as follows:

1) total number of points
2) total number of edges
3) total number of polygons
4) X, Y, Z values for all points
5) edge connect sequence for all edges
6) number of edges in a polygon
7) the edges that comprise this polygon
8) repeat 6 and 7 for each polygon

The total number of points, edges, and polygons are read and stored in their respective variable locations. The X, Y, Z values for all the points are read and stored in their respective arrays. The edge connect sequence data is stored in an array. For each polygon the number of edges and the edge numbers that comprise the polygon are read. If the first edge number for this polygon differs from the first edge of the edge connect sequence, the edge connect sequence and the first edge number for this polygon are sorted until they are the same.

Procedure FINDEDGE

This procedure is called by MAIN to read the edge numbers of a polygon. The edge number is returned if it has not been previously read. If the edge number was previously read the total edge count is incremented. The previous edge data is moved and its new location is
returned.

Procedure GETNEXTEDGE

This procedure is called by WARNOCK, INITPOLYGONS, and NOEDGE. The procedure returns the X, Y, Z values of the two end points of an edge.

Procedure GETZ

This procedure is called by WARNOCK. The procedure calculates the Z value of a point from the X, Y end point values.

Procedure ANGLE

This procedure is called by CLIP. The procedure determines the relationship of a point to the current window. The point window relationship is checked only if the point is outside of the window.

Procedure PUSH

This procedure is called by CLIP. The procedure determines the X or Y value of a point at the respective Y or X window boundary.

Procedure WCODE

This procedure is called by CLIP and PUSH. The procedure determines the relationship of a point to the current window. The point window relationship is checked whether the point is inside or outside of the window.
Procedure CLIP

This procedure is called by WARNOCK. The procedure determines if an edge passes through the window. The endpoints of an edge are decoded to determine their relative position to the window. If both endpoints are within the window, no further calculations are done. Otherwise each endpoint is coded as to whether it is above, below, left, right, or inside the window. If both endpoints are above, below, left, or right of the window a value is returned. No clipping takes place. If one endpoint is inside the window, the edge is clipped to either the X or Y window boundary.

Procedure INITPOLYGONS

The purpose of this procedure is to determine which polygon is closest to the viewer after the point and edge information has been read by MAIN. The procedure calculates the plane coefficients for each plane. The calculations are made from the first three end points of the polygon. The smallest Z value for each polygon is saved. If the polygon is perpendicular to the window (only an edge is visible) the polygon is deleted. The polygons are sorted according to their relative depths. The closest polygon is at the top of the list and the farthest at the bottom.

Procedure WARNOCK

The WARNOCK procedure looks at each window to determine if any edges intersect the window. Each polygon that does intersect
the window is flagged for later processing. After all the polygons are processed for window intersection, those polygons that do intersect the window are now checked to determine which polygon is closest to the viewer. The ranking value determined in the INITPOLYGONS procedure is used to determine the visible polygon(s). Those edges that are hidden by the surface of a polygon are eliminated from the list of polygons that intersect the window. When this process of determining the viewable polygon(s) is completed, the algorithm now must decide if the edges within the window can be drawn. If more than one polygon's edges intersect the window, the processing is terminated. The window is divided by two and the procedure calls itself to continue. When only one polygon intersects the window, the edges that intersect the window are output to the CALCOMP plotter. The edges output are clipped to the window's location in the display surface and to the window size. Clipping involves taking the end points of an edge and mathematically determining the intersection of the edge with the window. The portion of the edge that intersects the window is within the window boundaries is output to the CALCOMP plotter. This process continues until the entire display surface has been evaluated. At the conclusion of the evaluation the algorithm terminates execution.

Procedure NOEDGE

The purpose of this procedure is to determine if any edges of all the polygons possibly pass through the window. The procedure eliminates the window division of WARNOCK when no polygons exist
in a particular window size and location.